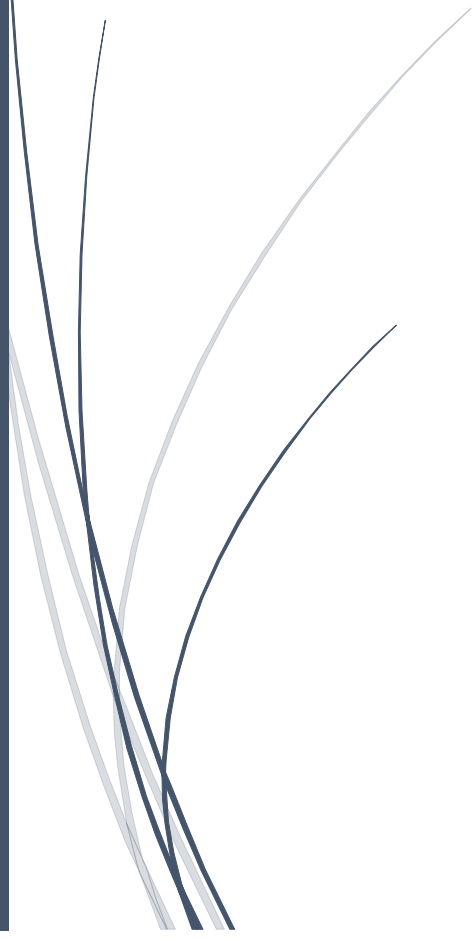
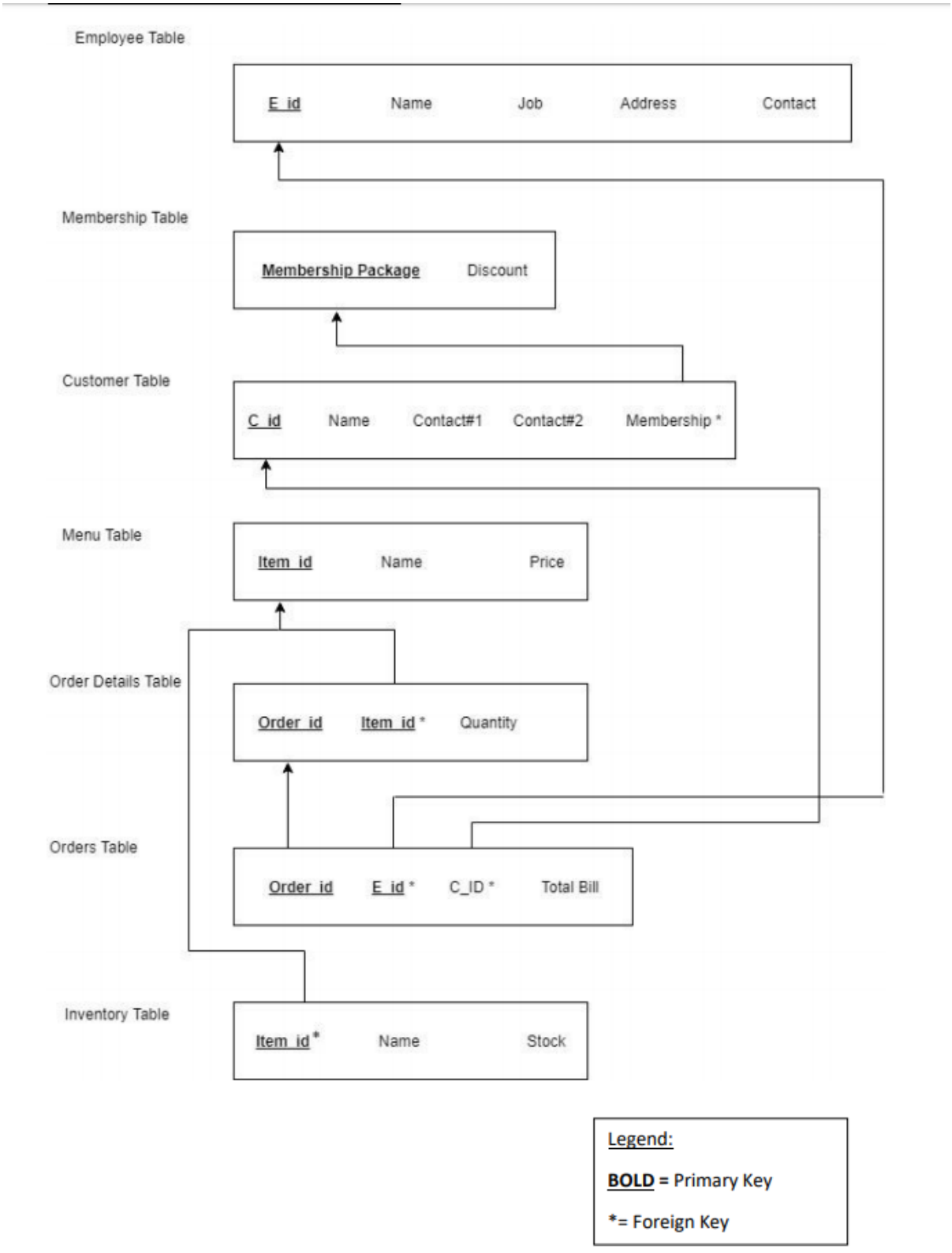
12/3/2018

RESTAURANT MANAGEMENT SYSTEM

20100006 Mohammad Raza Khawaja
20100056 Muhammad Hassan Arshad
20100108 Syed Hamza Ahmad

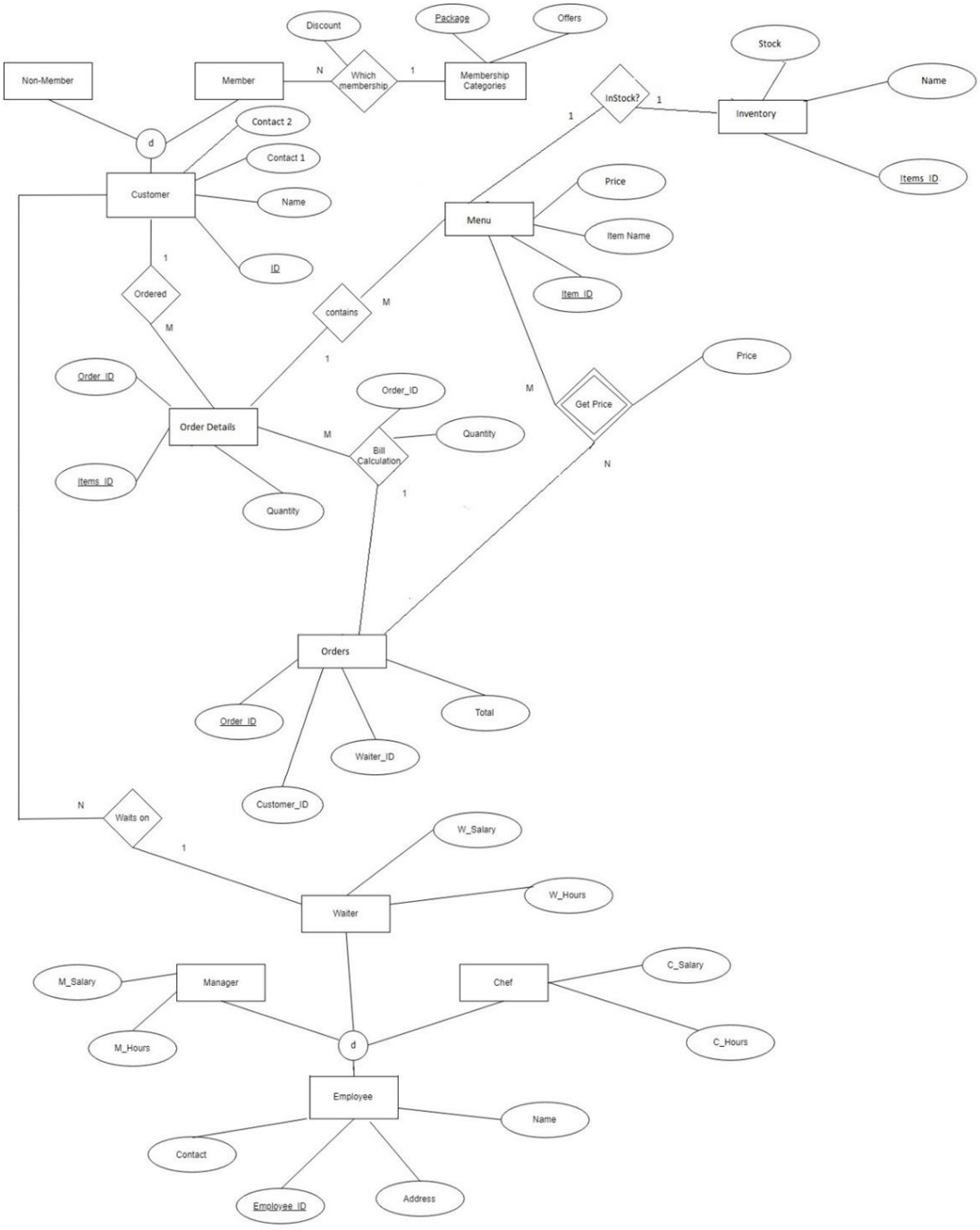


Updated Schema:



*** Assumption:** An employee will only have one contact number and address registered.

Updated EER Diagram:

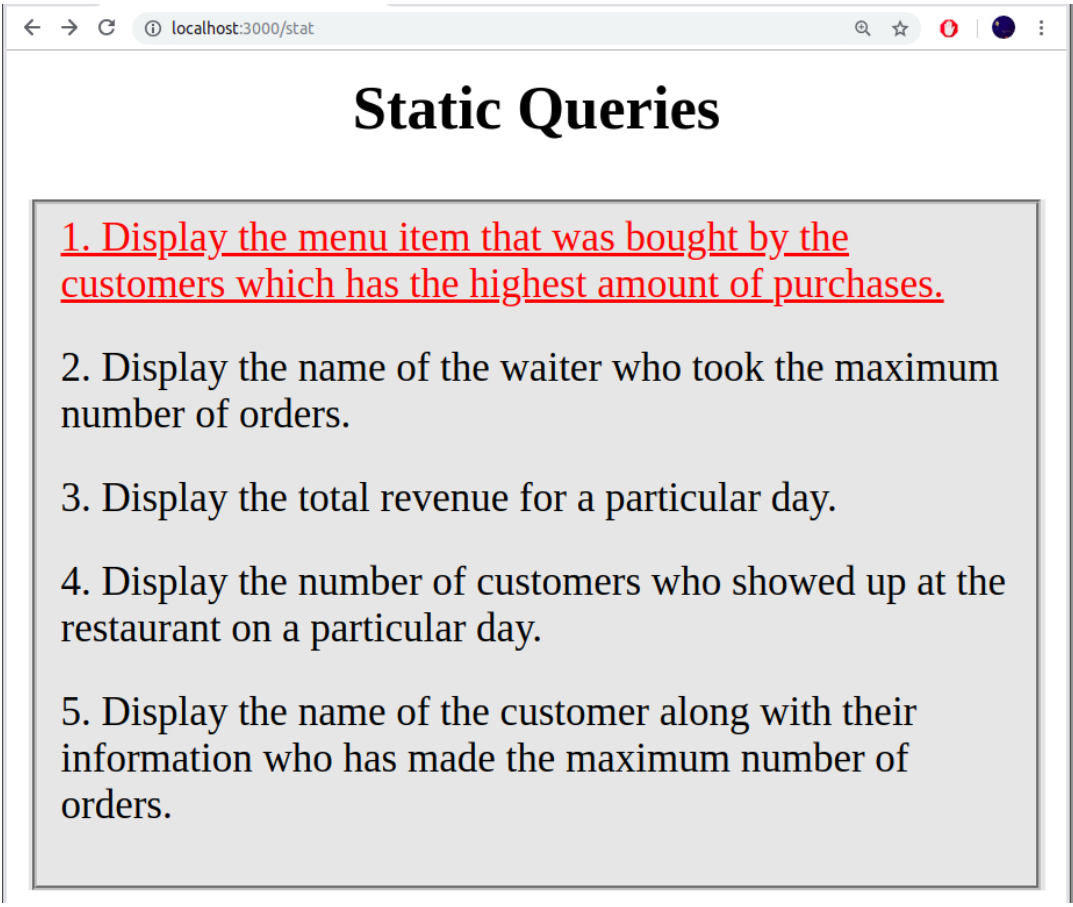


Queries Before Optimization:

For each query, the code (in JavaScript), its HTML input form, and its result is shown in the screenshots.

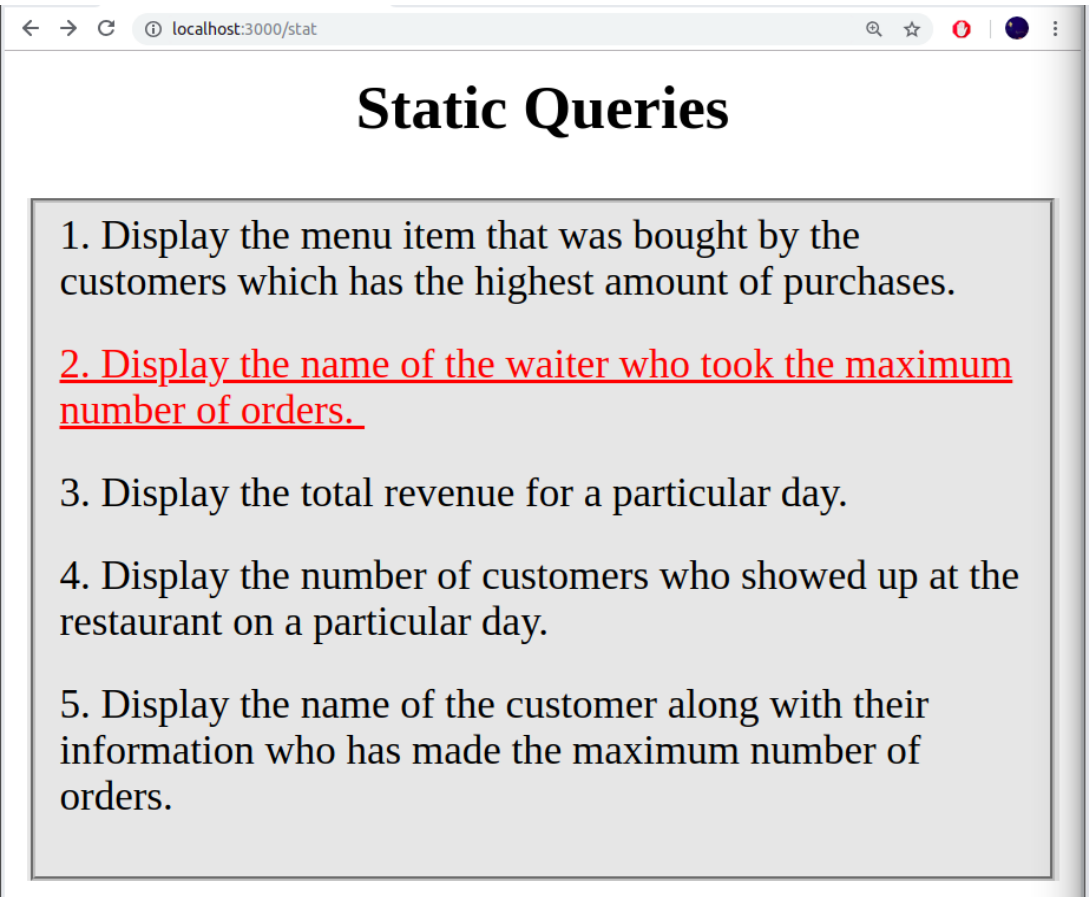
Static Query #1:

```
app.get('/sq1',function(req,res) {
    var max1;
    var max_id;
    var max_name;
    con.query("SELECT MAX(sq.quantity) as max FROM (SELECT item_id, SUM(quantity) AS quantity FROM order_details GROUP BY item_id) AS sq;",function(err,result) {
        max1 = result[0].max
        con.query("SELECT sq.item_id FROM (SELECT item_id, SUM(quantity) AS quantity FROM order_details GROUP BY item_id) AS sq WHERE sq.quantity = ?",max1,function(err,result) {
            max_id = result[0].item_id;
            con.query("SELECT item_name FROM menu WHERE item_id = ?",max_id,function(err,result) {
                max_name = result[0].item_name;
                res.send("Item purchased the most was: " + max_name);
            })
        })
    })
})
```



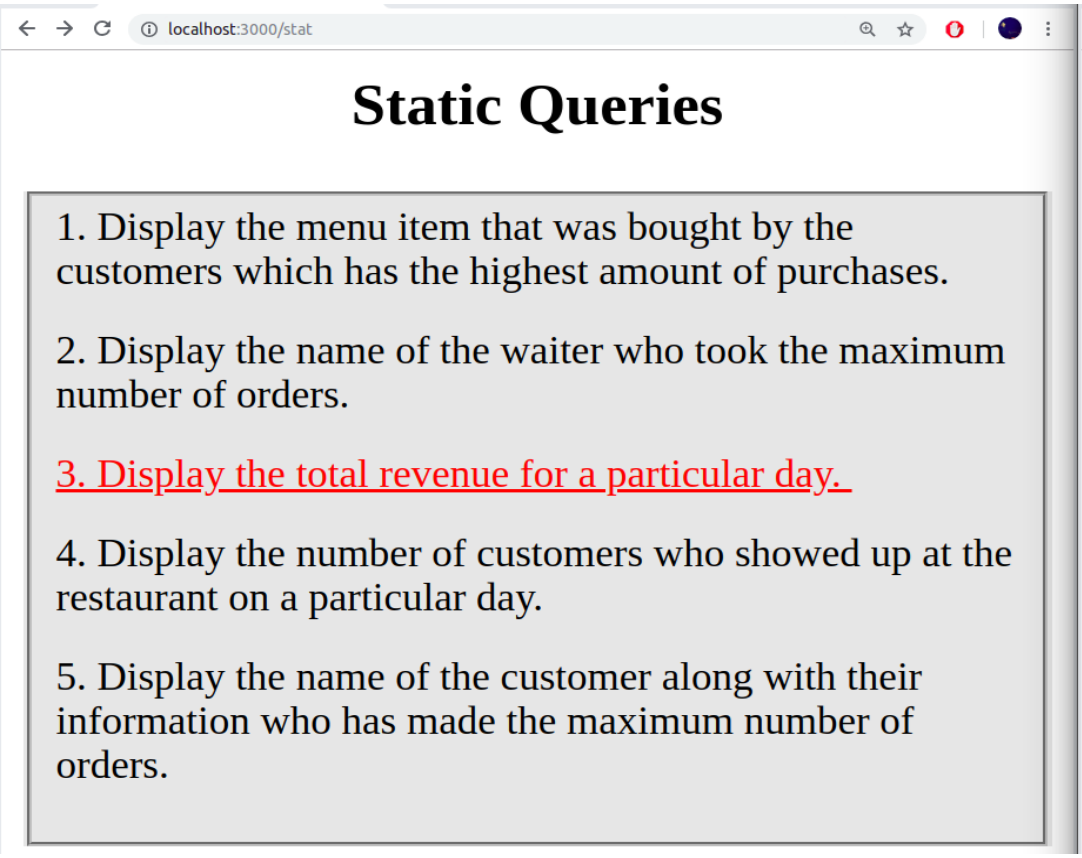
Static Query #2:

```
app.get('/sq2',function(req,res) {
    var wait_id;
    var wait_max;
    con.query("SELECT employee_id,COUNT(employee_id) AS waiter FROM orders GROUP BY employee_id ORDER BY waiter DESC LIMIT 1;", function(err,result){
        wait_id = result[0].employee_id
        con.query("SELECT name from employee WHERE employee_id = ?",wait_id,function(err,result) {
            wait_max = result[0].name;
            res.send("Waiter who took max orders: " +wait_max);
        })
    })

})
```



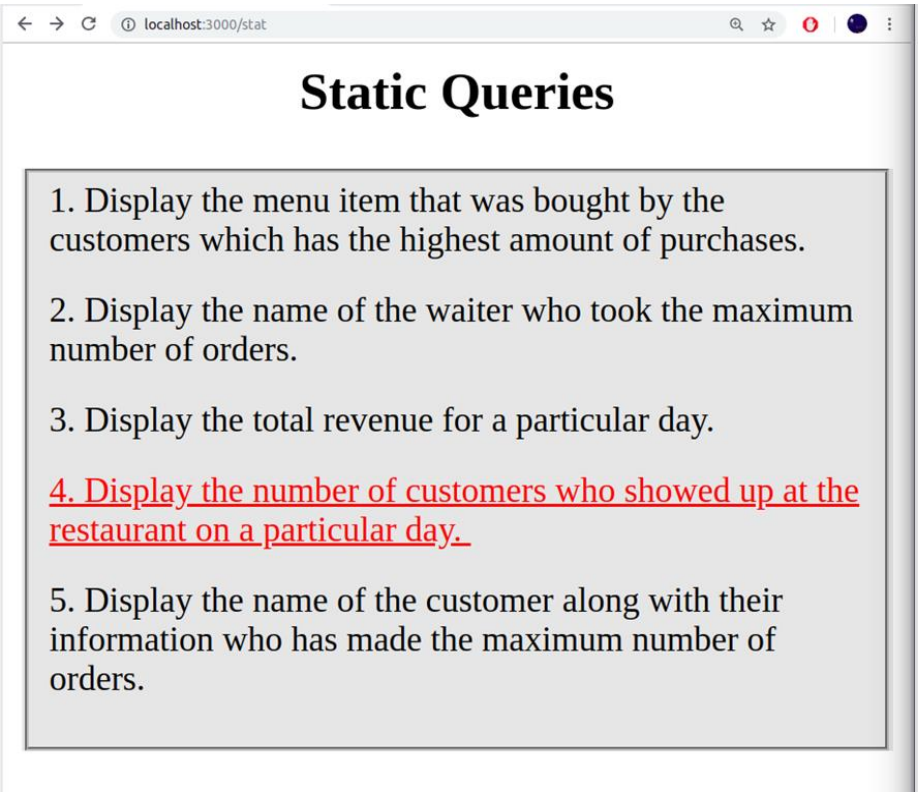
Static Query #3:

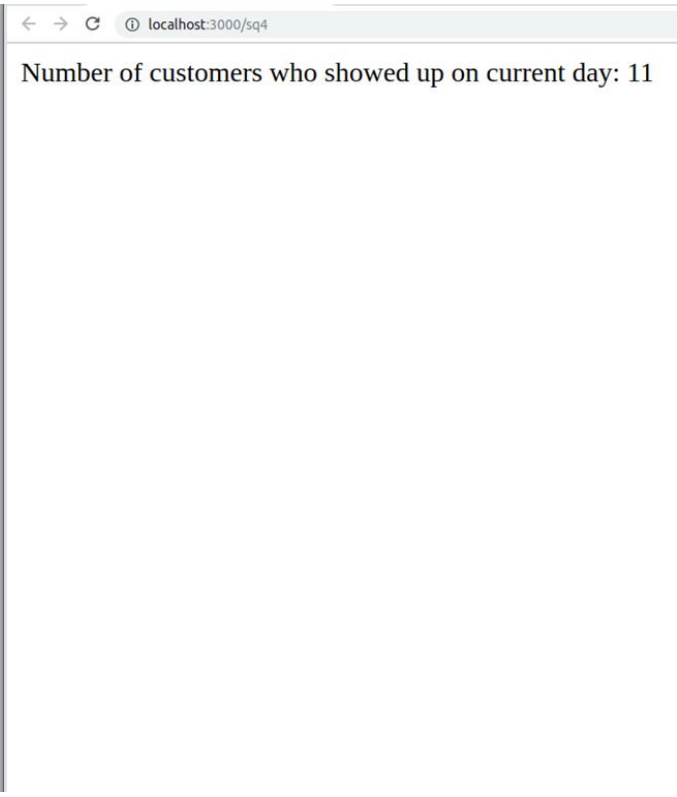
```
app.get('/sq3', function(req, res){
    con.query("SELECT SUM(total_bill) AS revenue FROM orders", function(err, result){
        res.status(200).send("Total Revenue for current day: " +(result[0].revenue).toString());
    })
})
```



Static Query #4:

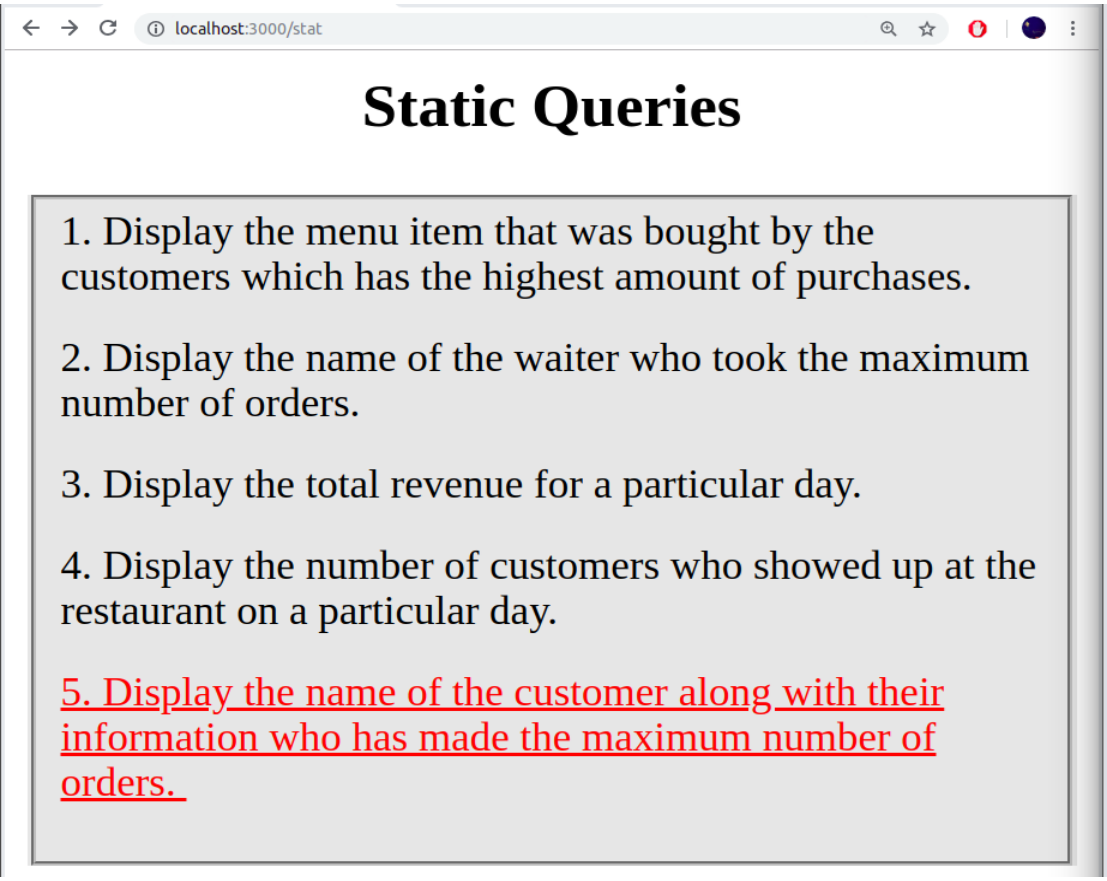
```
app.get('/sq4', function(req, res){
    con.query("create temporary table temp as select customer_id from orders;",function(err, result){
        con.query("select count(*) AS total from temp", function(err1,result1){
            con.query("drop table temp;", function(err2, result2){
                res.status(200).send("Number of customers who showed up on current day: " + (result1[0].total).toString());
            })

        })
    })

})
```

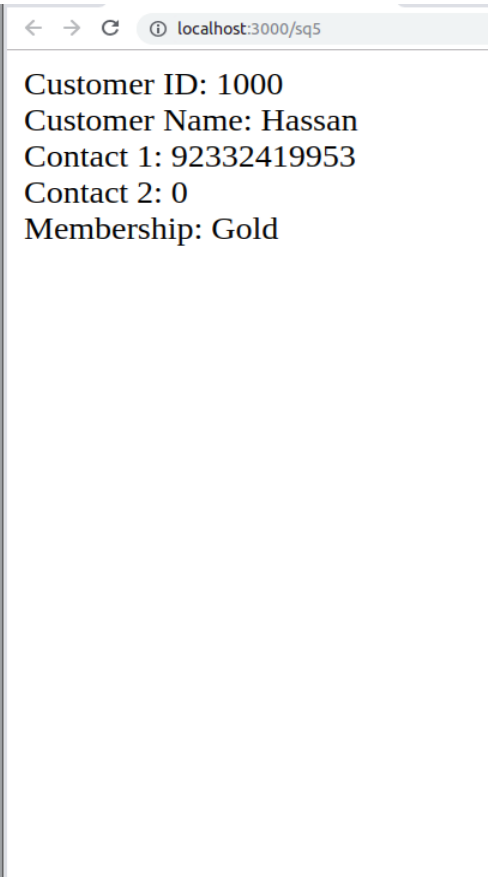




Static Query #5:

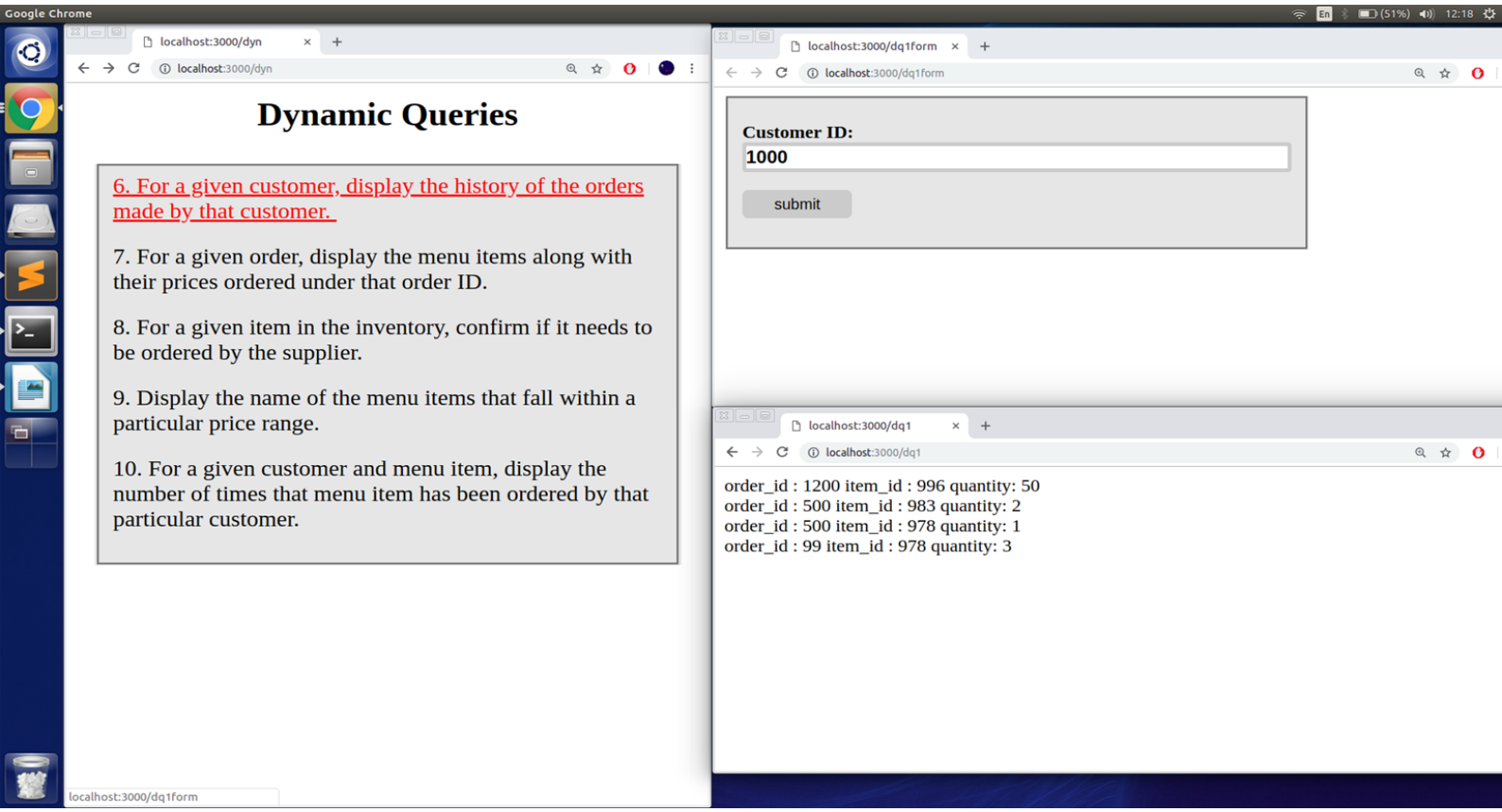
```
app.get('/sq5', function(req, res){
    var cust_id;
    var cust_max;
    con.query("SELECT customer_id,COUNT(customer_id) AS cust FROM orders GROUP BY customer_id ORDER BY cust DESC LIMIT 1;", function(err,result){
        cust_id = result[0].customer_id
        con.query("SELECT * FROM customer WHERE customer_id = ? ", cust_id, function(err1, result1){
            //res.status(200).send( (result1[0]));
            res.writeHead(200, { 'Content-Type': 'text/html' });
            res.write("Customer ID: " +result1[0].customer_id + '</br>' +
                "Customer Name: " + result1[0].name + '</br>' +
                "Contact 1: " + result1[0].contact1 + '</br>' +
                "Contact 2: " + result1[0].contact2 + '</br>' +
                "Membership: " + result1[0].membership
                );
            res.end('');
        })
    })

})
```





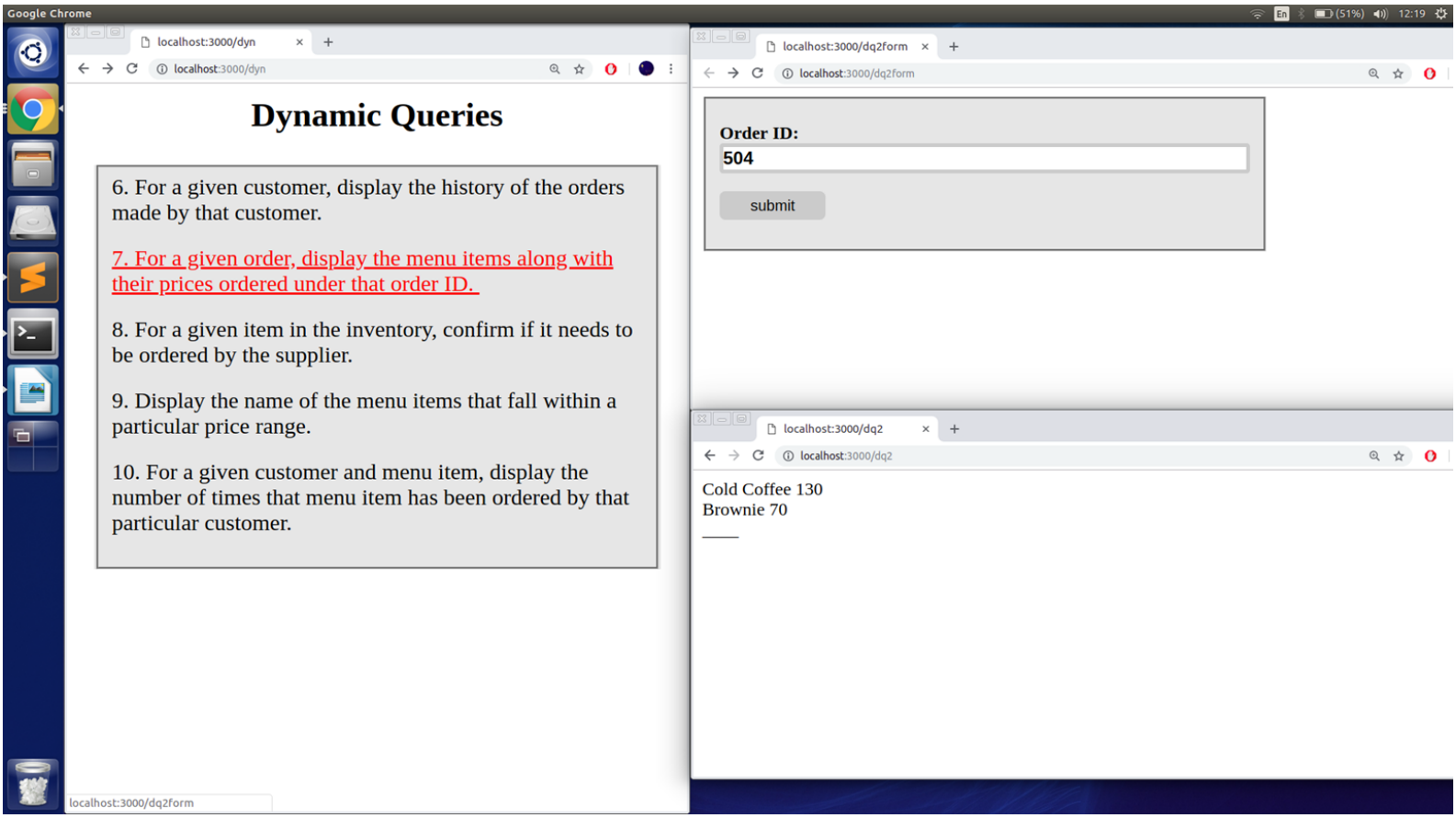
Dynamic Query #1:

```
app.post('/dq1', function(req, res){
    con.query("create temporary table temp as select * from orders WHERE customer_id = ?", parseInt(req.body.ID), function(err, result){
        con.query("SELECT order_details.order_id, order_details.item_id, order_details.quantity FROM order_details JOIN temp ON order_details.order_id=temp.order_id", function(err,result) {
            res.writeHead(200, { 'Content-Type': 'text/html' });
            for (var i = result.length - 1; i >= 0; i--) {
                res.write("order_id : " + result[i].order_id + " item_id : " + result[i].item_id + " quantity: " + result[i].quantity + "</br>");
            }
            res.end("");
        })
        con.query("drop table temp;",function(err,result) {
            if (result) {
                console.log("DROPPED")
            }
        });
    })
})
```



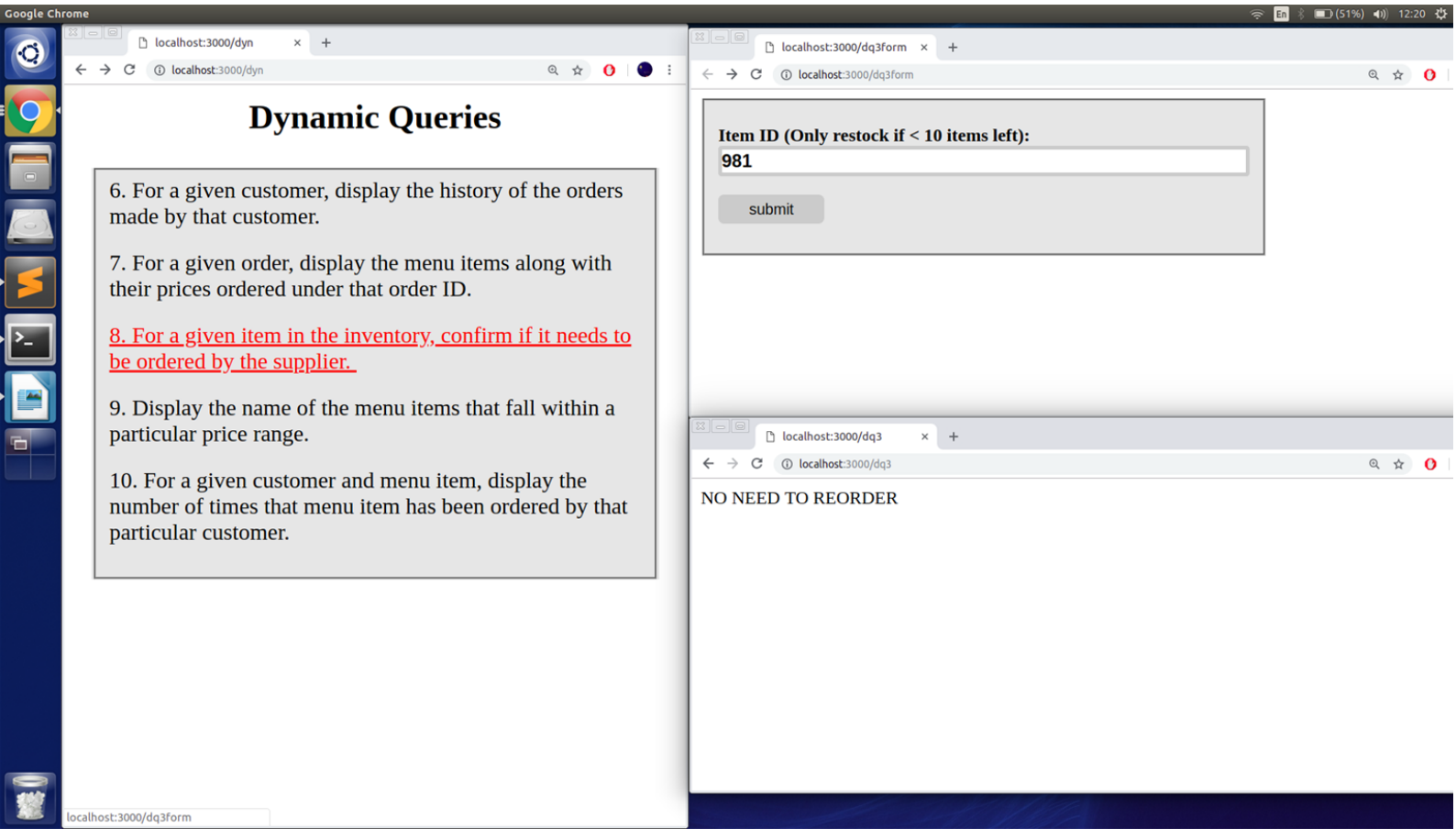
Dynamic Query #2:

```
app.post('/dq2', function(req, res){
    var i_d = parseInt(req.body.ID);
    con.query("create temporary table temp as select item_id from order_details WHERE order_id = ?",i_d,function(err,result) {
        con.query("SELECT menu.item_name, menu.price FROM menu JOIN temp ON menu.item_id=temp.item_id", function(err,result) {
            res.writeHead(200, { 'Content-Type': 'text/html' });
            for (i=0;i<length(result);i++)
            {
                res.write(result[i].item_name + "   " + result[i].price + '</br>');
            }
            res.end('____');
        })
        con.query("drop table temp;",function(err,result) {
            if (result)
                console.log("TABLE DROPPED");

        })
    })
})
```



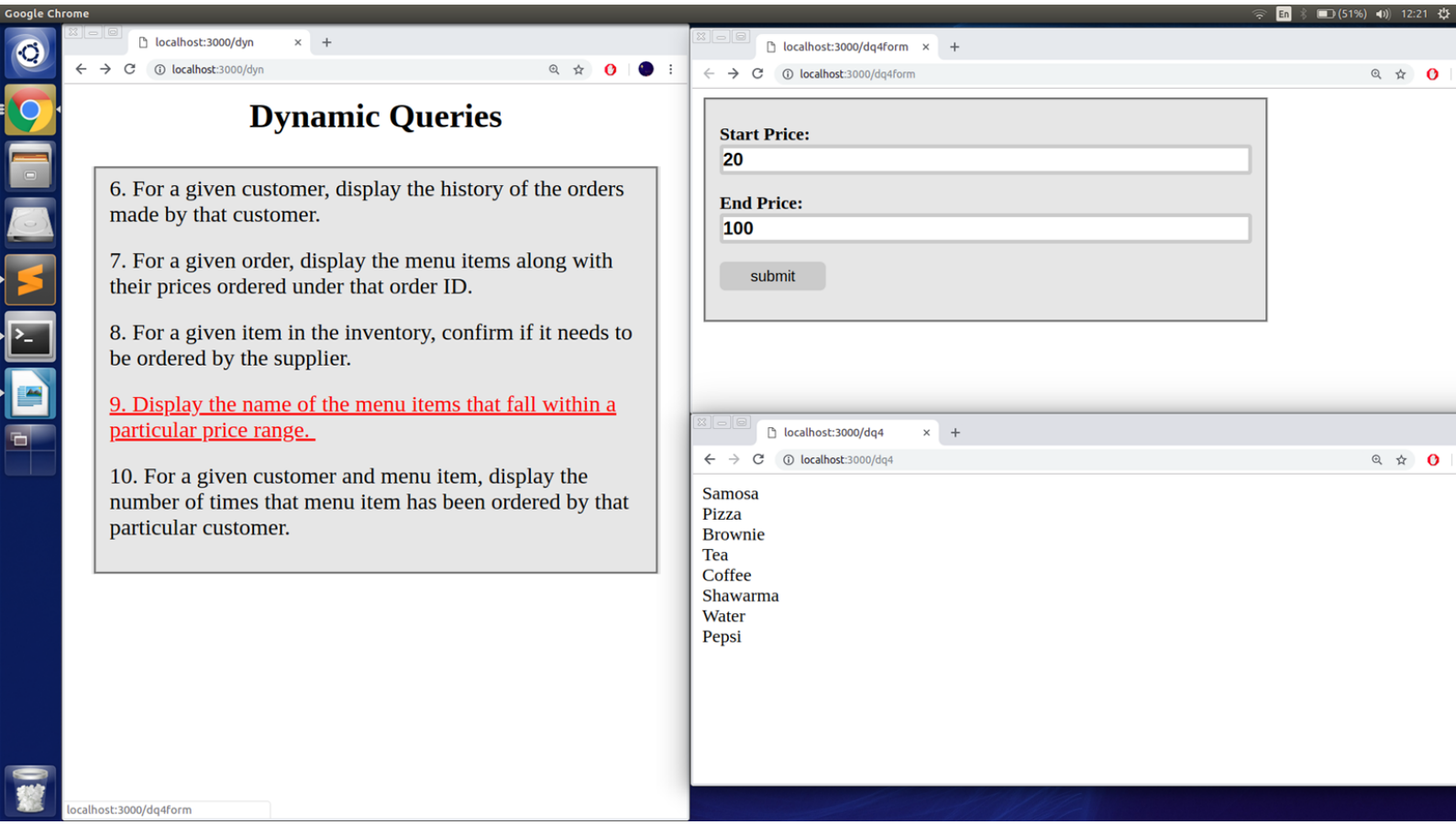
Dynamic Query #3:

```
app.post('/dq3', function(req, res) {
    var i_d = parseInt(req.body.ID);
    var b = 0;
    con.query("SELECT item_id FROM inventory WHERE stock < 10;",function(err,result){
        for (var i = result.length - 1; i >= 0; i--) {
            if (result[i].item_id == i_d) {
                b = 1;
            }
        }
        if (b) {
            res.send("NEEDS TO BE REORDERED");
        } else {
            res.send("NO NEED TO REORDER");
        }
    })
})
```



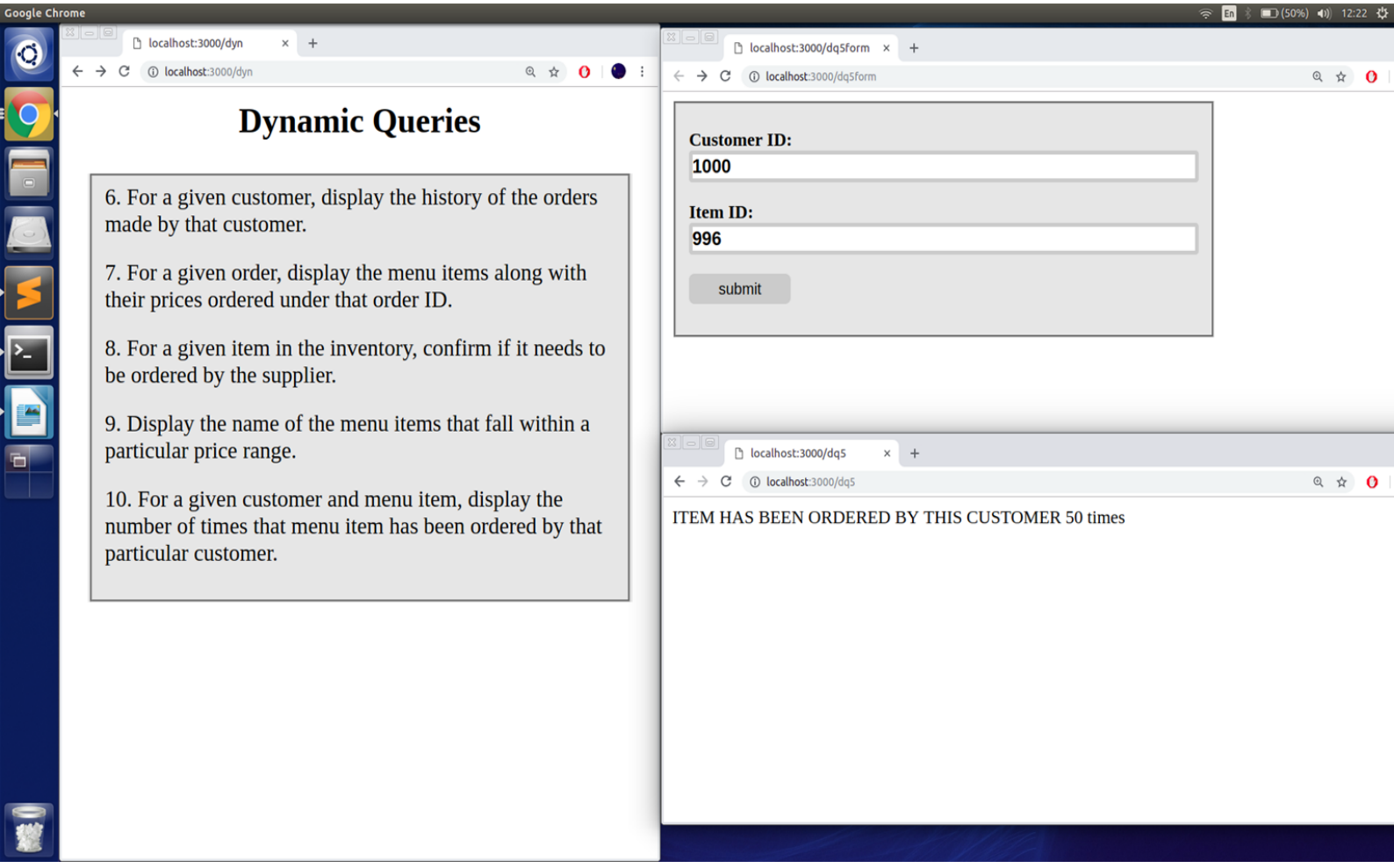
Dynamic Query #4:

```
app.post('/dq4',function(req,res) {
    var s;
    var e;
    s = parseInt(req.body.sprice);
    e = parseInt(req.body.eprice);

    con.query("SELECT item_name FROM menu WHERE price >= ? AND price < ?",[s,e],function(err,result) {
        res.writeHead(200, { 'Content-Type': 'text/html' });
        for (var i = result.length - 1; i >= 0; i--) {
            res.write(result[i].item_name + "</br>");
        }
        res.end("");
    })
})
```



Dynamic Query #5:

```
app.post('/dq5',function(req,res) {
    var cname = parseInt(req.body.cname);
    var iname = parseInt(req.body.iname);
    var i_d;
    con.query("create temporary table temp as SELECT customer_id FROM customer WHERE customer_id = ?",cname,function(err,result) {
        con.query("create temporary table temp1 as SELECT orders.order_id FROM orders JOIN temp ON orders.customer_id=temp.customer_id",function(err,result) {
            con.query("create temporary table temp2 as SELECT odd.order_id, odd.item_id, odd.quantity FROM order_details as odd JOIN temp1 ON odd.order_id=temp1.order_id",function(err,result){
                con.query("SELECT item_id, SUM(quantity) as quantity FROM temp2 GROUP BY item_id HAVING item_id = ?",iname,function(err,result) {
                    if (!length(result)) {
                        res.send("ITEM WAS NEVER ORDERED");
                    } else {
                    res.send("ITEM HAS BEEN ORDERED BY THIS CUSTOMER " + result[0].quantity + " times" );

                    }
                })
                con.query("drop table temp2");
            })

            con.query("drop table temp1");
        })
        con.query("drop table temp");
    })
})
```



Query Optimization:

Static Queries:

1. Three select operations, using callbacks for sub-queries in Node JS. All of them are O(n) in order. Hence, overall complexity is O(n). No optimization required.

2. Two select operations, first one selects in O(n) but also sorts in descending order, which comes to O(nlogn). So overall complexity is O(nlogn). Second is simple select statement is O(n). Total complexity is O(nlogn). No optimization required.

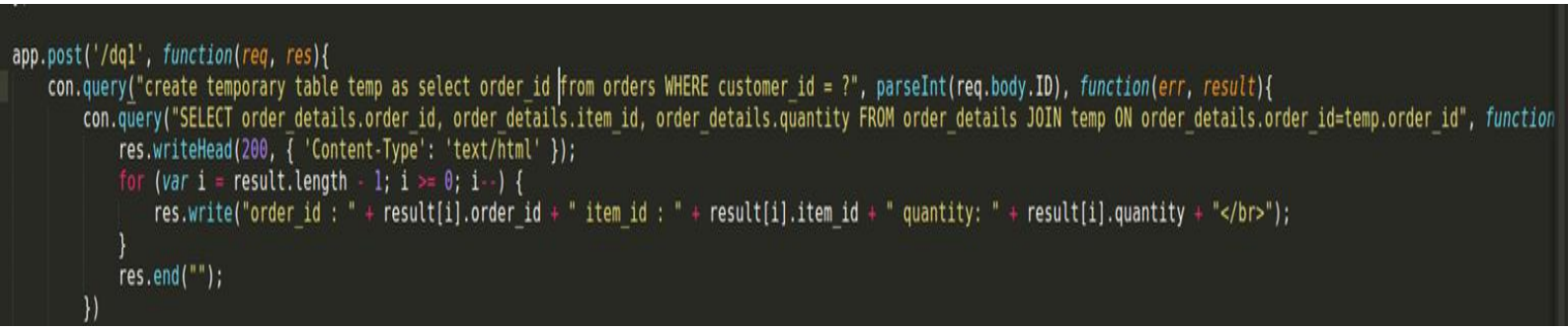
3. Select and sum operations, which are in O(n). We do not use select * to avoid retrieving any data that will not be used. No optimization required.

4. We have already used a temporary table to store results instead of view, which is the better choice when accessing results from multiple queries. First select query takes O(n) and the second query is also in O(n). We do not use select * so we do not get any unnecessary data.

5. First select operation uses descending order sorting in O(nlogn). The second select operation takes O(n), hence the final complexity is O(nlogn). Using a LIMIT statement prevents taxing the production database with a large query, only to find out the query needs editing or refinement.

Dynamic Queries:

1. A temporary table is created, where we originally selected all fields from the orders table. But joining this would increase the complexity so instead, we only selected the order id's from the orders table in the temporary table and then used the join operation on this temp table. This temp table being smaller in size will improve our complexity. The join operation causes the time complexity to be $O(n^2)$. Join statements prove to be an improvement over correlated queries.



```
app.post('/dq1', function(req, res){
    con.query("create temporary table temp as select order_id from orders WHERE customer_id = ?", parseInt(req.body.ID), function(err, result){
        con.query("SELECT order_details.order_id, order_details.item_id, order_details.quantity FROM order_details JOIN temp ON order_details.order_id=temp.order_id", function
            res.writeHead(200, { 'Content-Type': 'text/html' });
            for (var i = result.length - 1; i >= 0; i--) {
                res.write("order_id : " + result[i].order_id + " item_id : " + result[i].item_id + " quantity: " + result[i].quantity + "</br>");
            }
            res.end("");
        })
```

2. Already optimized since unlike dynamic query 1, we did not select all the fields from the order_details table, and only selected the field needed which was the item_id, so space and time complexity both are reduced. The join operation causes the time complexity to be $O(n^2)$.

3. A select operation is used, which is in order O(n). We already do not use select *, so we do not receive any unwanted data. Hence, no need to optimize.

4. A select operation on menu table is used, using the given range. Hence, no need to optimize this query either.

5. First operation is a temporary table creation using select operation which is O(n). This temporary table is joined with another temporary table which is in $O(n^2)$, but we are only selecting the fields that are required, hence it is already optimized. The third operation is also join, which joins the result with a new temporary table and again, only the relevant fields are selected before joining, to reduce complexity. Final operation is a simple select statement, using Group By which is of O(n). The overall complexity is $O(n^2)$.